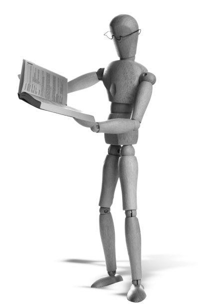
open source library

Michael Kofler
Bernd Öggl

PHP 5.3 & MySQL 5.1

Grundlagen, Programmiertechniken, Beispiele



18 Zeiterfassung

Nicht nur Selbstständige stehen vor der Aufgabe, ihre geleistete Arbeitszeit zu erfassen und zu dokumentieren, auch Angestellte können mit einem effizienten Zeitmanagement ihren Arbeitserfolg steigern. Die klassische Stechuhr ist mit der zunehmenden Mobilität der arbeitenden Menschen überfordert. Ein modernes Zeiterfassungssystem sollte online erreichbar sein und möglichst auch mit einem Mobiltelefon bedient werden können. Aus diesen Gründen liegt es nahe, eine Software auf der Basis von PHP und MySQL im Netzwerk zu verwenden.

Die hier vorgestellte Lösung bietet die Funktionen einer elektronischen Stechuhr, die über einen Internet-Browser aktiviert wird. Vorhandene Einträge können korrigiert werden, und neue Einträge können manuell hinzugefügt werden. Im betrieblichen Alltag müssten solche Änderungen autorisiert werden. Auf die dazu notwendige Benutzerverwaltung wurde hier verzichtet, was das System in dieser Form hauptsächlich für Einzelpersonen interessant macht. Auf die notwendigen Erweiterungen für den Mehrbenutzerbetrieb wird im letzten Abschnitt hingewiesen. Einige Funktionen des Programms sind:

- Monatslisten der erbrachten Stunden, gruppiert nach Projekten
- Export der Einträge nach CSV, XML und iCalendar (zum dynamischen Einbauen in Outlook, Google-Kalender, ...)
- Eine Pause-Funktion, die einen unkomplizierten Zeitabzug für jede Arbeitseinheit bietet
- Hierarchische Projektverwaltung mit Trigger-Funktionen der Datenbank
- Monitor zum Überwachen der Online-Aktivität
- Grafische Darstellung der Arbeitszeit auf einer interaktiven Zeitachse

18.1 Bedienung

Verwaltung von Projekten

Jedes Projekt ist eine eigenständige Einheit, und die Arbeitszeit wird getrennt erfasst. Über die Funktion PROJEKTE – ANZEIGEN/BEARBEITEN lassen sich Projekte thematisch strukturieren und benennen. Hier kann man Unterprojekte einfügen und Projekte löschen, sofern sie keine weiteren Unterprojekte besitzen.

Zeiterfassung starten

Auf der Startseite der Anwendung befindet sich ein Auswahlmenü mit einer Liste von bereits eingerichteten Projekten. Um die Zeiterfassung zu starten, muss ein Projekt (Task) ausgewählt werden, dem die Arbeitszeit zugewiesen ist.

Abbildung 18.1: Die Startseite der Zeiterfassung

Sobald die Zeiterfassung aktiviert wird, besteht die Möglichkeit, ein Statusfenster zu öffnen. So lange dieses Fenster geöffnet bleibt, wird in der Datenbank die Online-Aktivität festgehalten. Vergisst man, die Zeiterfassung zu beenden, so kann die letzte Online-Zeit beim späteren Editieren des Eintrags ein nützlicher Hinweis auf das Arbeitsende sein (wird der Browser beendet, stoppt die Update-Funktion). Im laufenden Betrieb hat es sich bewährt, ein solches Fenster minimiert geöffnet zu halten, da ich häufig vergesse *auszustempeln*.

Einträge ansehen und bearbeiten

Die gearbeiteten Zeiteinheiten können in einer Tabelle ausgegeben werden, wahlweise nach Monaten oder nach Projekten gruppiert.

Will man einen Eintrag bearbeiten, um etwa eine Pause einzufügen, so klickt man auf die Dauer des Eintrags und kommt zu einer neuen Seite (Abbildung 18.3).

Wird das Datum oder die Uhrzeit verändert, überprüft der Browser automatisch, ob es sich um eine gültige Zeitspanne handelt, und errechnet die neue Dauer. Auf dieser Seite können Einträge auch gelöscht werden.

open source library

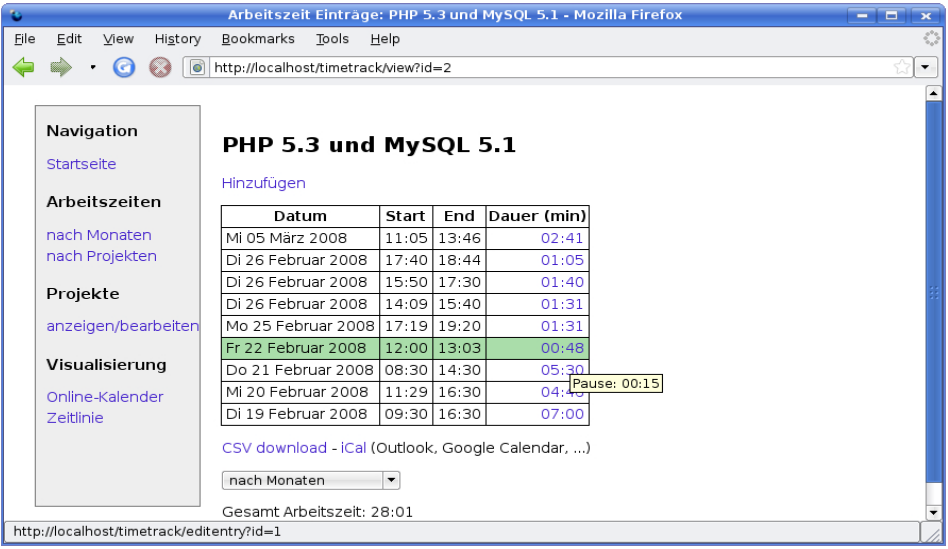


Abbildung 18.2: Liste aller Arbeitseinheiten für ein Projekt

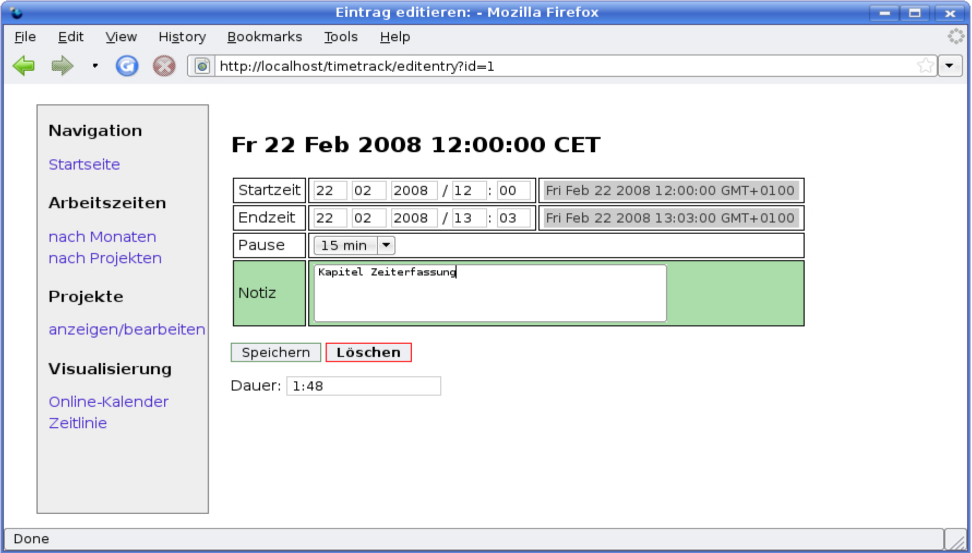


Abbildung 18.3: Bearbeiten einer Arbeitseinheit

Visualisierung

Damit die mühevoll erbrachte Arbeitszeit auch grafisch ansprechend dargestellt werden kann, sieht das Programm unterschiedliche Formate für den Export vor. Mit Hilfe des *iCalendar*-Formats kann die Arbeitszeit in Microsoft Outlook oder ähnlichen Programmen (Korganizer, Novell-Evolution, ...) angezeigt werden. Die

open source library

Einträge können dabei entweder in eine Datei exportiert und statisch angezeigt werden oder, noch eleganter, dynamisch als *Internet-Kalender* eingebunden werden.

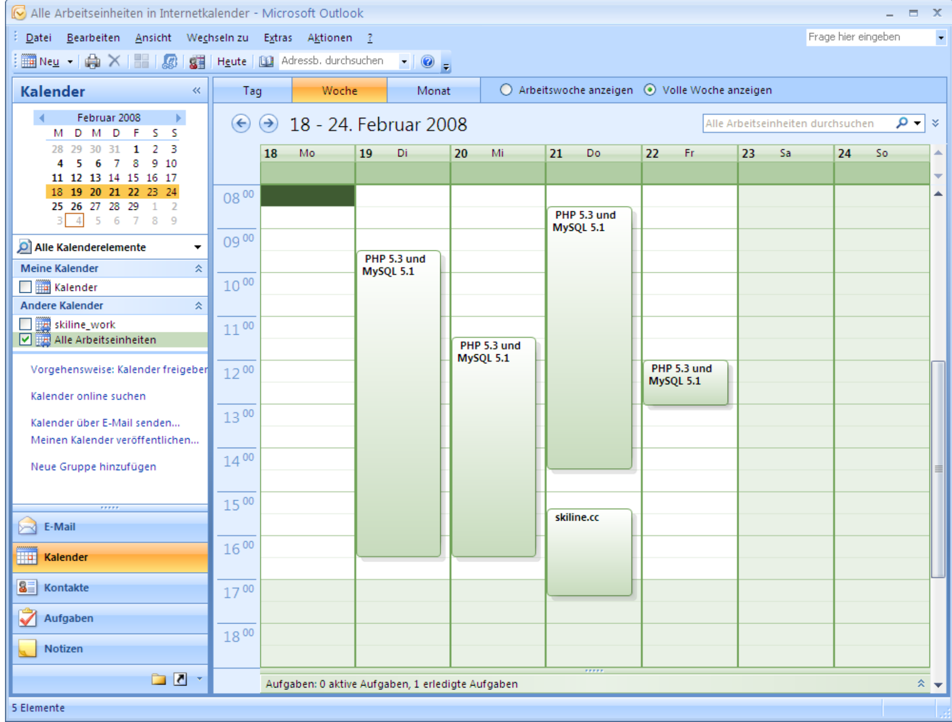


Abbildung 18.4: Die Arbeitszeit als Online-Kalender in Microsoft Outlook

Eine weitere Möglichkeit der Anzeige bietet die *Zeitlinie*, wobei die Einträge auf einer skalierbaren Zeitachse angezeigt werden. Zur Darstellung dient eine Bibliothek aus dem SIMILE-Projekt, das am Massachusetts Institute of Technology (MIT) vorangetrieben wird. Die Bibliothek ist frei verfügbar und wird dynamisch über das Internet geladen.

Die Zeitlinie gliedert sich in drei Bereiche: Die Monatsleiste steht ganz unten, darüber sehen Sie die Tage, und der größte Teil im oberen Bereich zeigt die Stundeneinteilung eines Tages. Die Arbeitseinheiten werden in allen drei Bereichen eingetragen und zeigen dementsprechend längere oder kürzere Balken. Beim Anklicken eines Eintrags in der Stundenleiste werden Informationen zu diesem Eintrag und zum ganzen Projekt angezeigt (siehe Abbildung 18.5).

Außerdem besteht für jedes Projekt noch die Möglichkeit, es als CSV (Komma getrennte ASCII-Datei) zu exportieren. Das Format eignet sich zum Öffnen in einem Tabellenkalkulationsprogramm (Excel oder OpenOffice Calc) oder zum Import in eine andere Datenbank.

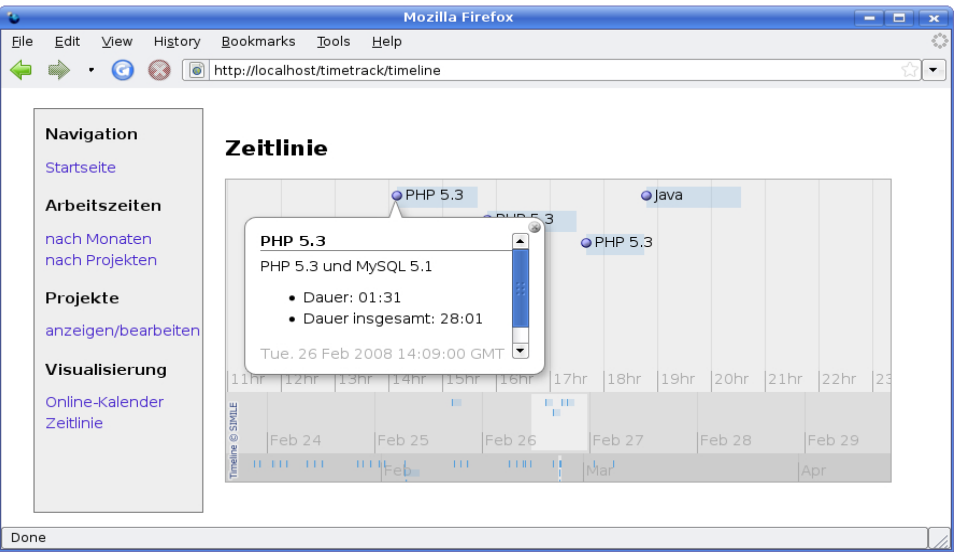


Abbildung 18.5: Darstellung der Arbeitszeit auf der Zeitlinie des SIMILE-Projekts

2.csv - OpenOffice.org Calc

Datei Bearbeiten Ansicht Einfügen Format Extras Daten Fenster Hilfe

E9 =SUMME(B9-A9-C9)

	A	B	C	D	E	F	G
1	#Start	End	Pause	ProjektName	Summe		
2	22.02.08 12:00	22.02.08 13:03	00:15:00	PHP 5.3 und MySQL 5.1	00:48		
3	21.02.08 08:30	21.02.08 14:30	00:30:00	PHP 5.3 und MySQL 5.1	05:30		
4	20.02.08 11:29	20.02.08 16:30	00:15:00	PHP 5.3 und MySQL 5.1	04:46		
5	19.02.08 09:30	19.02.08 16:30	00:00:00	PHP 5.3 und MySQL 5.1	07:00		
6	25.02.08 17:19	25.02.08 19:20	00:30:00	PHP 5.3 und MySQL 5.1	01:31		
7	26.02.08 14:09	26.02.08 15:40	00:00:00	PHP 5.3 und MySQL 5.1	01:31		
8	26.02.08 17:40	26.02.08 18:44	00:00:00	PHP 5.3 und MySQL 5.1	01:04		
9	26.02.08 15:50	26.02.08 17:30	00:00:00	PHP 5.3 und MySQL 5.1	01:40		
10					23:50		
11							

Tabelle1

Tabelle 1 / 1 Standard 100% STD Summe=01:40

Abbildung 18.6: Der CSV-Import in OpenOffice Calc

18.2 Datenbankdesign

Das Schema der *timetrack*-Datenbank enthält fünf Tabellen. Zeiteinträge (die *timelog*-Tabelle) werden dabei getrennt von den Projekten (*task*-Tabelle) verwaltet. Die Anwendung sieht vor, dass Zeiteinträge mit einer Notiz versehen werden können. Damit in der zentralen *timelog*-Tabelle nicht leere Felder ohne eine Notiz enthalten sind, wird diese Information in eine eigene Tabelle (*note*) ausgelagert. Gleiches trifft auch auf mögliche Pausen in einer Arbeitseinheit zu (Tabelle *break*).

Das digitale Gegenstück zur Stempelkarte ist die *timelog*-Tabelle. Hier werden die Start- und Endzeit der Arbeitseinheit als DATETIME-Datentyp festgehalten. Ein zusätzliches Feature ist die *lastseen*-Spalte: Solange ein Browser-Fenster auf die Statusseite zeigt, wird in dieser Spalte alle drei Minuten der *TIMESTAMP*-Wert aktualisiert. Wird der Computer, von dem aus die Arbeitszeit gestartet wurde, ausgeschaltet, endet die Aktualisierung.

Datenbanktechnisch interessant stellt sich die hier verwendete Lösung der hierarchischen Projektverwaltung dar (Tabelle *task* und *tasktree*). Während die Hierarchien in Abschnitt 12.7 hauptsächlich durch PHP-Programmierung aufgelöst werden, wird hier versucht, möglichst viel Arbeit in der Datenbank zu erledigen. Dazu enthält die Tabelle *tasktree* eine vollständige Liste aller Projektnummern mit den übergeordneten Projekten und der Verschachtelungstiefe. Glücklicherweise kümmert sich das Datenbanksystem selbst um das Befüllen und Leeren der Tabelle: Die Trigger-Funktionen in der *task*-Tabelle übernehmen diesen Part.

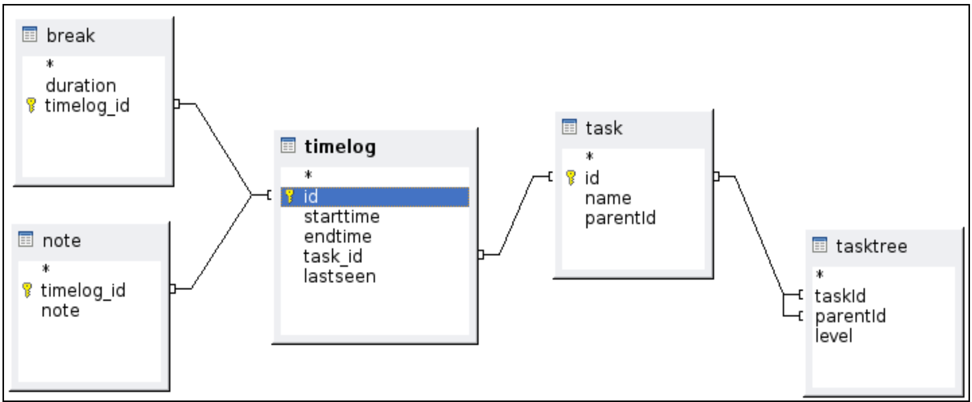


Abbildung 18.7: Das Schema der timetrack-Datenbank

Trigger-Funktionen

Folgende Überlegung steckt hinter der *tasktree*-Tabelle: Wenn jedes Projekt mit seinem übergeordneten Projekt und der Verschachtelungstiefe in einer Tabelle abgespeichert ist, ist es möglich, in einer SQL-Abfrage alle Eltern oder alle Kindelemente zu ermitteln. Nach jedem Einfügen oder Löschen eines Projekts muss die *tasktree*-Tabelle aktualisiert werden. Hier kommt die Trigger-Funktion ins Spiel, die für ein neues Projekt folgendermaßen aussieht:

open source library

```
DELIMITER ;;
CREATE TRIGGER treetrigger
AFTER INSERT ON task
FOR EACH ROW
  BEGIN
    INSERT INTO tasktree(taskId, parentId, level) VALUES (NEW.id, NEW.id, 0);
    INSERT INTO tasktree(taskId,parentId,level)
      SELECT NEW.id,tasktree.parentId,tasktree.level+1
        FROM task,tasktree
        WHERE NEW.parentId = tasktree.taskId
          AND tasktree.taskId = task.id;
  END
DELIMITER ;
```

Nach dem INSERT in der Tabelle *task* wird die Funktion für jede eingefügte Zeile ausgeführt. Zu Beginn wird die soeben erstellte Projektnummer (der Primärschlüssel der *task*-Tabelle) als *taskId* und *parentId* mit dem Level 0 eingetragen. Der Zugriff auf das *AUTO_INCREMENT*-Feld erfolgt über *NEW.id*, wobei sich das *NEW* auf den gerade eingefügten Datensatz in der task Tabelle bezieht. Jetzt wird es ein bisschen komplizierter: In der *tasktree*-Tabelle wird die neue *taskId* für alle hierarchischen Ebenen mit dem entsprechenden *level* eingetragen. Dazu werden in der *tasktree*-Tabelle alle übergeordneten Projekte gesucht und mit der neuen *taskId* und dem *level*-Feld um einen Wert erhöht (*tasktree.level+1*) eingetragen.

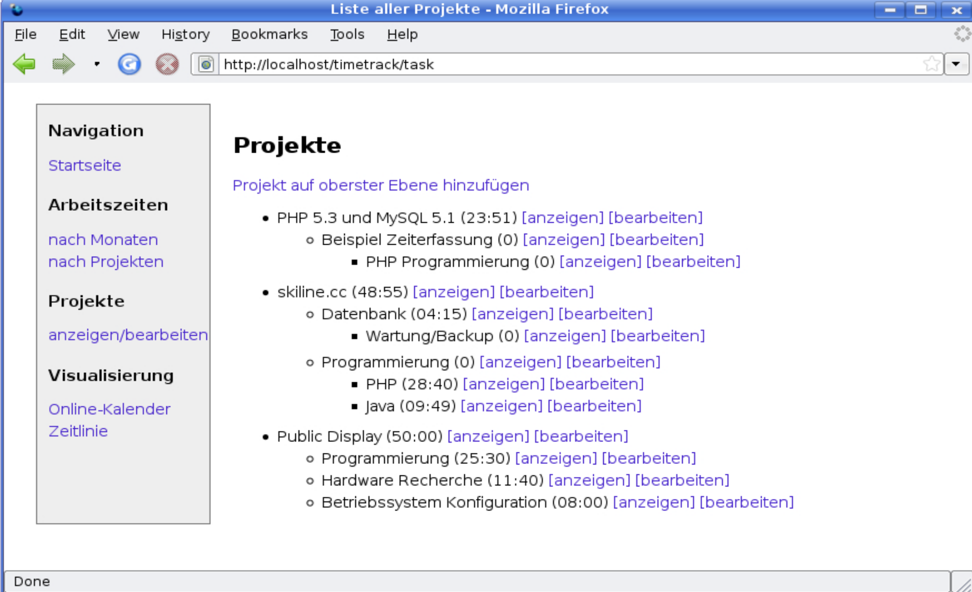


Abbildung 18.8: Hierarchische Liste der Projekte

Zur Erklärung hier ein Beispiel: Das Projekt *Beispiel Zeiterfassung* wird als Unterprojekt von *PHP 5.3 und MySQL 5.1* eingefügt.

```
SELECT * FROM task
```

id	name	parentId
2	PHP 5.3 und MySQL 5.1	0
3	Beispiel Zeiterfassung	2

Die Trigger-Funktion fügte in der *tasktree*-Tabelle daraufhin zwei Zeilen ein:

```
SELECT * FROM tasktree WHERE taskId = 3
```

taskId	parentId	level
3	3	0
3	2	1

Für das Unterprojekt *PHP-Programmierung* (ID 17) fügt die Trigger-Funktion drei Zeilen in der *tasktree*-Tabelle ein:

```
SELECT * FROM tasktree WHERE taskId = 17
```

taskId	parentId	level
17	17	0
17	3	1
17	2	2

Entscheidend dabei ist das SELECT-Kommando in der Trigger-Funktion: Es werden alle Einträge aus der *tasktree*-Tabelle für das übergeordnete Projekt selektiert und mit der neuen Projekt-ID eingetragen. Der Wert in der *level*-Spalte wird jeweils um einen Zähler erhöht.

Die Trigger-Funktion zum Löschen der Einträge ist vergleichsweise einfach:

```
DELIMITER ;;
CREATE TRIGGER deltree
  AFTER DELETE ON task
  FOR EACH ROW
  BEGIN
    DELETE FROM tasktree WHERE taskId = OLD.id;
  END ;;
DELIMITER ;
```

Nachdem der Eintrag aus der *task*-Tabelle gelöscht worden ist, werden auch alle Einträge, bei denen die *taskId* der gerade gelöschten ID entspricht (*OLD.id*), aus der *tasktree*-Tabelle entfernt.

Aber wozu der ganze Aufwand mit der *tasktree*-Tabelle? Durch sie wird es möglich, mit einer SQL-Abfrage alle in der Hierarchie darüber liegenden Elemente zu erfragen:

open source library

```
SELECT p.*,t.level FROM task n, tasktree t, task p
WHERE n.id = ? AND n.id = t.taskId
  AND p.id = t.parentId ORDER BY t.level
```

Die *task*-Tabelle muss zweimal eingebunden werden. Zuerst werden (mit dem Alias *n*) in der *tasktree*-Tabelle alle Einträge auf ein Projekt beschränkt (*n.id = ? AND n.id = t.taskId*). Dann werden aus dieser Liste die übergeordneten Projekt-IDs genommen und mit *task*-Tabelle (Alias *p*) verschnitten (*p.id = t.parentId*). Die Sortierung nach dem *level*-Feld liefert eine korrekte Liste aller übergeordneten Projekte.

Eine Liste aller Unterprojekte zu generieren ist etwas schwieriger: Die Baumstruktur kann sich noch weiter verästeln, und die korrekte Sortierung ist hier nicht mehr gegeben:

```
SELECT c.*,tt.level FROM task t, tasktree tt, task c
WHERE t.id = ? AND t.id = tt.ParentId
  AND tt.taskId = c.id AND c.id <> ? ORDER BY tt.level
```

Die hier gezeigte Variante von Baumstrukturen in der Datenbank beherrscht nur das Einfügen neuer Elemente und das Löschen des letzten Elements einer Hierarchie. Eine vollständige Implementierung müsste noch das Verschieben von Elementen vorsehen. Da diese Funktionalität auch in der Benutzeroberfläche nicht vorgesehen ist, wird hier auf die *UPDATE*-Trigger-Funktion verzichtet.

Hinweis

Zur Darstellung von Hierarchien in der Datenbank finden Sie in diesem Buch drei Vorschläge (Kapitel 12.7 mit PHP-Unterstützung, Kapitel 14.6 mit Stored Procedures und dieses Kapitel mit Triggern). Eine Internet-Recherche nach *trees in SQL* wird Ihnen zeigen, dass sich viele Menschen mit dem Thema beschäftigen und dass es unterschiedlichste Lösungen gibt.

Die hier vorgestellte Lösung ist den beiden vorangegangen unterlegen, da sie das Problem der Sortierung der Kindelemente nicht zufriedenstellend löst. In diesem Kapitel wurde nicht vorrangig auf Hierarchien Wert gelegt. Die Idee zu diesem Ansatz stammt aus einer Internet-Recherche:

http://www.codeproject.com/KB/database/Trees_in_SQL_databases.aspx

18.3 Aufbau des Codes

Der Code verteilt sich auf die folgenden Unterverzeichnisse:

includes/	Verzeichnis, in dem alle Klassen liegen
modules/	Verzeichnis der *Module*, die die Applikationslogik steuern
script/	Javascript-Dateien für die Zeitlinie und die Datumskorrektur
style/	Stylesheets für die Standardansicht und für den Ausdruck
maintenance/	Struktur der Tabellen

open source library

Auf der untersten Verzeichnisebene liegen nur vier Dateien:

index.php	der Front-Controller
config.inc.php	Konfigurationseinstellungen
bootstrap.php	Standard-Einstellungen (*error_reporting*) und globale Variablen
.htaccess	Apache-Anweisungen für den Front-Controller

Folgende Klassen finden sich im includes-Verzeichnis:

config.php	Konfiguration der Anwendung
dateformatter.php	Statische Methoden zur Datumsausgabe
db.php	Kapselung aller Datenbankzugriffe
domlist.php	DOM-Funktionen für die hierarchische Projektverwaltung
error.php	Statische Funktionen zur Fehlerausgabe
htmlpage.php	filtert Formulareingaben, gibt HTML-Kopf- und -Fußzeile aus
icalendar.php	formatiert die Zeiteinträge im iCalendar-Format
timetrack.php	erzeugt eine Liste aller (aktiven) Projekte
timetrackentry.php	repräsentiert eine Arbeitseinheit
timetracktask.php	repräsentiert ein Projekt

Die Dateien im modules-Verzeichnis können direkt im Browser aufgerufen werden.

add.php	fügt eine Arbeitseinheit hinzu
view.php	zeigt Arbeitseinheiten an
editentry.php	editiert eine Arbeitseinheit
addtask.php	fügt ein Projekt hinzu
edittask.php	ändert ein Projekt
index.php	die Startseite der Anwendung
ical.php	Ausgabe der Zeiteinträge im iCalendar-Format
month.php	Ausgabe der Arbeitseinheiten für ein Monat
task.php	hierarchische Anzeige der Projekte
timeline.php	Anzeige der Arbeitseinheiten auf der Zeitlinie
viewcsv.php	Anzeige der Arbeitseinheiten im CSV-Format
xmldata.php	generiert den XML-Code für die Zeitlinie
heartbeat.php	Überwachung der Online-Aktivität
setalive.php	aktualisiert den Zeitstempel für die letzte Online-Aktivität

Das Beispiel wurde durchgänig mit einem objektorientieren Ansatz entworfen. Jede Klasse wurde in einer eigenen Datei gespeichert, was zu der großen Anzahl von Dateien führt. In PHP wäre es auch möglich, alle Klassen in nur eine Datei zu speichern, mir erschien diese Aufteilung aber übersichtlicher.

Wie schon in Kapitel 3 erwähnt wurde, eignet sich Objektorientierung vor allem für große Programme. Das vorliegende Beispiel ist nicht groß, aber es bietet die Möglichkeit, an vielen Stellen erweitert zu werden. Ein Zeiterfassungssystem muss oft an spezielle Bedürfnisse angepasst werden. Der vorliegende Code kann als

Kernstück verwendet werden, wobei sich Erweiterungen durch die Objektorientierung leicht anknüpfen lassen.

Der Ausdruck *Module* bezieht sich im Folgenden auf kleine Code-Stücke, die die Logik der Anwendung steuern. Das *task*-Modul kann zum Beispiel Projekte anzeigen und bietet die Links zum Hinzufügen/Ändern und Löschen von Projekten. Sobald man eine neue PHP-Datei im modules-Verzeichnis anlegt, findet der Front-Controller das Modul und führt es auf Anfrage aus. Beim Erstellen eines neuen Moduls braucht man sich nicht um das HTML-Gerüst zu kümmern. Ein Hello-World-Modul kommt mit drei Zeilen Code aus:

```
<?php  // Beispieldatei helloworld.php
$html->printHeader("Hello World");
print "Hello World.";
```

Der Front-Controller kümmert sich um die Initialisierung der *HtmlPage*-Klasse, HTML-Header, die Navigationsleiste und den HTML-Abschluss. Beim Aufruf von http://localhost/timetrack/helloworld erhalten Sie eine gültige XHTML 1.0-Webseite.

18.3.1 Front-Controller

Der Code im vorliegenden Beispiel verwendet das Konzept eines Front-Controllers. Dabei werden alle Seitenaufrufe auf ein einziges PHP-Skript umgeleitet, ohne dass der Anwender davon erfährt (die URL ändert sich dabei nicht). Der Vorteil dieses Ansatzes ist, dass Einstellungen, die bei jedem Seitenaufruf gemacht werden (zum Beispiel das Herstellen der Datenbankverbindung oder die korrekten *LOCALE*-Einstellungen) nicht in jedem Skript aufgerufen werden müssen.

Umleitung mit mod_rewrite

Damit alle Anfragen an den Front-Controller weitergeleitet werden, braucht man die Unterstützung des Webservers. Der Apache Webserver unterstützt die *Rewrite*-Anweisungen erst nach dem Laden des entsprechenden Moduls (*mod_rewrite*). Sollte das Modul bei Ihrem System nicht geladen sein, hilft folgende Zeile in der Apache-Konfiguration (mehr Informationen zum Laden von Modulen finden Sie in Abschnitt 1.4):

```
LoadModule rewrite_module modules/mod_rewrite.so        # für Linux
```

Die Regeln, wie der Seitenaufruf umgeschrieben wird, stehen in einer .htaccess-Datei (mehr Informationen zu .htaccess-Dateien finden Sie in Abschnitt 1.4). Dabei wird zuerst die *RewriteEngine* gestartet (andernfalls ignoriert der Webserver die Anweisungen).

```
# .htaccess-Einstellungen für den Front-Controller
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php?m=$1 [L,QSA]
```

Bevor die *RewriteRule* alle Anfragen an das index.php-Skript umleitet, wird mit der *RewriteCond*-Anweisungen überprüft, ob eine gewünschte Datei oder ein Verzeichnis vorhanden ist. Die Variable *%{REQUEST_FILENAME}* enthält den vollen Pfad- und Dateinamen der angefragten URL. Durch das Ausrufungszeichen (*!*) wird die Umleitung abgebrochen, wenn eine Datei (*-f*) oder ein Verzeichnis (*-d*) existiert. Würde man diese Regeln nicht einsetzen, könnte das Programm keine Bilder, Stylesheets oder externen Javascript-Dateien vom Server laden, da auch diese Anfragen auf das PHP-Skript umgeleitet würden.

Die *RewriteRule*-Anweisung funktioniert folgendermaßen:

```
RewriteRule Suchmuster Ersetzung [Option,Option,...]
```

Das Suchmuster ist dabei ein regulärer Ausdruck (mehr zu regulären Ausdrücken in Abschnitt 5.9). Bei der Ersetzung kann auf ein gefundenes Suchmuster (das innerhalb der runden Klammern gespeichert ist) über die Referenz auf *$1*, *$2*, ... zurückgegriffen werden. Im vorliegenden Beispiel wird die gesamte Zeichenkette gesucht (*^(.*)$*) und dem Skript index.php per *GET* mit dem Schlüssel *m* übergeben. Die Option *L* (für *last rule*) bewirkt, dass bei einer erfolgreichen Ersetzung keine weiteren Ersetzungen mehr durchgeführt werden. *QSA* steht für *query string append*, wodurch eine zusätzliche Kontrolle der zu ersetzenden Werte ausgelöst wird.

Nehmen wir an, Sie laden die Zeitlinie im Programm: Die Adresse im Webbrowser lautet http://localhost/timetrack/timeline. Wie Sie in der Logdatei des Webservers sehen können, lautet die Anfrage folgendermaßen:

```
"GET /timetrack/timeline HTTP/1.1"
```

Da die *RewriteRule* in der .htaccess-Datei im Verzeichnis timetrack steht, gilt für die Anfrage nur der Teil hinter dem timetrack-Verzeichnis. Die *RewriteRule* tritt in Kraft, da es hier weder ein Verzeichnis noch eine Datei timeline gibt. Das von dem regulären Ausdruck gesuchte Muster lautet *timeline*-und es kommt zu folgender Ersetzung: *index.php?m=timeline*

Das *mod_rewrite* wird als das Schweizermesser der URL-Behandlung bezeichnet. Es kann ungemein hilfreich sein, leider aber auch sehr kryptisch und undurchschaubar. Dokumentation zu diesem Modul findet man im Internet ausreichend, trotzdem bleibt es oft ein bisschen *Voodoo*, wie die Apache-Webseite meint. Zur Fehlersuche empfehlen sich die Konfigurationseinstellungen *RewriteLog* und *RewriteLogLevel*.

open source library

Module laden

Ab diesem Zeitpunkt kommt der Front-Controller zum Einsatz. Er lädt die Datei bootstrap.php, in der globale Einstellungen zurzeitzone und zum Fehlerverhalten (*error_reporting*) gesetzt werden. Weiters wird in der *__autoload*-Funktion festgelegt, in welchem Verzeichnis die zu ladenden Klassen liegen (*__autoload* wird in Abschnitt 3.2.8 näher erläutert).

```
<?php  // Front Controller: index.php
require_once('bootstrap.php');
$html = new HtmlPage($_REQUEST);
$module = $html->getStringFromRequest('m');
if (!$module) {
  $module = 'index';
}
// Dateien sind kleingeschrieben und haben keine Sonderzeichen:
$module = preg_replace("/[^a-z0-9-]/", "", strtolower($module));
$moduleFile = dirname(__FILE__) . '/modules/' . $module . '.php';
if (file_exists($moduleFile)) {
  require($moduleFile);
} else {
  Error::pageNotFound();
}
if ($html->footLine) {
  $html->printFooter();
}
```

Die Hauptaufgabe des Front-Controllers ist es, die gewünschte Seite zu laden. Nachdem der Wert für den Schlüssel *m* extrahiert worden ist (*getStringFromRequest* aus der *HtmlPage*-Klasse), wird versucht, eine entsprechende Datei im modules-Verzeichnis zu finden. Mit der *require*-Funktion wird der Inhalt dieser Datei geladen. Sollte die Datei nicht existieren, wird aus der statischen *Error*-Klasse die Funktion *pageNotFound* aufgerufen. Neben dem Text, dass die Seite nicht existiert, wird dabei der HTML-Statuscode 404 im Header gesetzt. Ohne diese Anweisung lässt sich bei einer späteren Analyse der Logdateien nicht feststellen, ob eine Seite gefunden wurde oder nicht.

Die *require*-Anweisung ist sicherheitstechnisch nicht ganz unbedenklich: Da die Funktion auch die Möglichkeit hat, Dateien aus dem Internet zu laden, könnte ein Angreifer versuchen, hier ein bösartiges Skript einzuschleusen. Im vorliegenden Code wird das durch die explizite Angabe der Basis (*dirname(__FILE__)*) verhindert. Da man beim Umgang mit Benutzer-Eingaben aber nicht vorsichtig genug sein kann, wurde hier mit dem *preg_replace*-Kommando eine noch striktere Einschränkung vorgenommen: Module dürfen nur Kleinbuchstaben, Zahlen und das Minus-Zeichen enthalten. So wird wird das Dateisystem vor neugierigen Zugriffen geschützt.

18.3.2 Die Klassen für Projekte und Arbeitseinheiten

Um dem objektorientierten Ansatz, der für dieses Beispiel gewählt wurde, zu entsprechen, wurden eigene Klassen für die wichtigsten Elemente eingerichtet. Die zentralen Objekte bei der vorgestellten Zeiterfassung sind Projekte und Arbeitseinheiten. Dementsprechend gibt es eine Klasse für die Arbeitszeit (*TimetrackEntry*) und eine Klasse für ein Projekt (*TimetrackTask*).

TimetrackTask

Die *TimetrackTask*-Klasse repräsentiert ein definiertes Projekt (im Code als *task* bezeichnet). Im Konstruktor der Klasse werden die Informationen zum Projekt inklusive der Gesamtdauer abgefragt.

```
SELECT p.*,
    SUM(TIME_TO_SEC(TIMEDIFF(t.endtime,t.starttime))-COALESCE(b.duration,0))
      AS totalTime
  FROM task p LEFT JOIN timelog t ON t.task_id = p.id
    LEFT JOIN break b ON b.timelog_id = t.id
  WHERE p.id = ?
  GROUP BY p.id
```

Bei der Berechnung der Gesamtdauer ist zu berücksichtigen, dass jede dem Projekt zugewiesene Zeiteinheit (Tabelle *timelog*) mit einer Pause versehen sein kann. Die Tabelle *break* wird daher mit einem *LEFT JOIN* an die Abfrage angehängt. Alle Einträge ohne Pausen liefern jetzt einen *NULL*-Wert in der Spalte für die Pausendauer. Da dieser *NULL*-Wert einen Fehler in der Zeitberechnung auslösen würde, wird er beim Bilden der Summe durch die Zahl 0 ersetzt (*COALESCE(b.duration,0)*). Das Fragezeichen in der *WHERE*-Anweisung wird später durch das *prepared*-Statement mit der ID des gewünschten Projekts ersetzt.

Die Klasse enthält auch eine statische Methode, um ein neues Projekt zu erzeugen. Die *factory*-Funktion benötigt als Parameter den Projektnamen und die ID des übergeordneten Projekts (0 für Projekte auf der obersten Ebene).

```
public static function factory($n, $parentId) {
  $dbHandle = Db::getHandle();
  $sql = "INSERT INTO task (name,parentId) VALUES (?, ?)";
  $stmt = $dbHandle->prepare($sql);
  $stmt->bind_param("si", $n, $parentId);
  return $stmt->execute();
}
```

Durch die Verwendung von *prepared*-Statements braucht man sich hier keine Sorgen wegen einer SQL-Injection zu machen. Beim Binden der Parameter an das Statement wird dem Interpreter mitgeteilt, dass die Variable *$n* vom Typ String (*s*) und *$parentId* vom Typ Integer (*i*) ist. Das korrekte Quotieren der Werte übernimmt PHP.

open source library

Weitere Methoden in dieser Klasse dienen zum Abfragen der Projekte, die in der Hierarchie darüber beziehungsweise darunter liegen (vergleiche Abschnitt 18.2). Außerdem verfügt die Klasse über Methoden, um alle oder einzelne Arbeitseinträge abzufragen, die diesem Projekt zugewiesen sind.

```
public function getEntry($id) {
  $sql = "SELECT *,UNIX_TIMESTAMP(starttime) AS start_sec, "
    . " UNIX_TIMESTAMP(endtime) AS end_sec FROM timelog "
    . " WHERE task_id = ? AND id = ?";
  $stmt = $this->dbHandle->prepare($sql);
  $stmt->bind_param("ii", $this->id, $id);
  $e =  $this->dbHandle->queryObjectArrayPrepared($stmt);
  return new TimetrackEntry((int)$e[0]->id, (int)$e[0]->start_sec,
    (int)$e[0]->end_sec, $this->id);
}
```

Der Rückgabewert dieser Funktion ist ein neues Objekt vom Typ *TimetrackEntry*. Es repräsentiert eine Arbeitseinheit.

TimetrackEntry

Im Gegensatz zum *TimetrackTask* wird beim Erzeugen eines neuen *TimetrackEntry*-Objekts kein Datenbankzugriff benötigt. Bei Projekten mit vielen Arbeitseinheiten würde die Datenbank unnötig stark belastet. Dem Konstruktor müssen daher die wichtigsten Merkmale des Objekts übergeben werden.

```
public function __construct($id, $s, $e, $task_id) {
    $this->id = $id;
    $this->start = $s;
    $this->end = $e;
    if ($this->end == NULL || $this->end == 0) {
        $this->end = time();
        $this->unfinished = TRUE;
    }
    $this->task_id = $task_id;
}
```

Start- und Endzeit werden als UNIX_TIMESTAMP, also in Sekunden nach dem 1.1.1970 übergeben (vergleiche Abschnitt 10.2). Außerdem braucht das neue Objekt noch die ID des Projekts, dem es zugeordnet ist (*$task_id*), und die ID des eigenen Eintrags (*$id*).

Die *TimetrackEntry*-Klasse bietet unter anderem Funktionen, um die Dauer eines Eintrags zu ermitteln (abzüglich einer möglichen Pause), ihn als HTML-Tabelle auszugeben, den Eintrag zu löschen, einen neuen Eintrag zu erzeugen oder einen veränderten Eintrag zu speichern.

```
public function save() {
  $sql = "REPLACE INTO timelog (id,starttime,endtime,task_id) "
    . " VALUES (?,?,?,?) ";
  $dbHandle = Db::getHandle();
  $stmt = $dbHandle->prepare($sql);
  $stmt->bind_param("issi", $this->id,
    strftime("%Y-%m-%d %H:%M:%S", $this->start),
    strftime("%Y-%m-%d %H:%M:%S", $this->end), $this->task_id);
  $stmt->execute();
  $ins_id = $stmt->insert_id;
  if ($this->note) {
    $sql = "REPLACE INTO note (timelog_id, note) "
      . " VALUES (?,?)";
    $stmt = $dbHandle->prepare($sql);
    $stmt->bind_param("is", $ins_id, $this->note);
    $stmt->execute();
  }
  if ($this->removeTime > 0) {
    $sql = "REPLACE INTO break (duration, timelog_id) "
      . " VALUES (?,?)";
    $stmt = $dbHandle->prepare($sql);
    $stmt->bind_param("ii", $this->removeTime, $ins_id);
    $stmt->execute();
  }
}
```

Praktisch erweist sich in diesem Abschnitt die *REPLACE INTO*-Funktion von MySQL. Ist eine Notiz für ein Projekt schon vorhanden, wird sie mit dem neuen Wert überschrieben, und gab es bisher keinen Eintrag, so wird eine neue Zeile in der Tabelle *note* angelegt. Gleiches gilt für die Tabelle *break*, die mögliche Pausen verwaltet.

18.4 Wichtige Codefragmente

18.4.1 Datenbankverbindung mit der Klasse Db

Die *Db*-Klasse leitet sich direkt von der von PHP zur Verfügung gestellten *mysqli*-Klasse ab. Das hat einerseits den Vorteil, dass man sie wie die *mysqli*-Klasse verwenden kann, andererseits wird sie noch um einige Funktionen bereichert, die das Arbeiten etwas erleichtern.

Im Konstruktor der Datenbank-Klasse wird die Verbindung mit dem MySQL-Server hergestellt und die Datenbank ausgewählt – eine bereits bekannte Prozedur. Neu ist die erweiterte Fehlerbehandlung, wenn die Verbindung nicht zustande kommen kann. Durch das Abfangen des Verbindungsfehlers (*catch*) und das Auswerten der Fehlernummer (*$e->getCode()*) können konkrete Hinweise zur Fehlerbehebung ausgegeben werden.

open source library

```
try {
  parent::__construct($mysqlhost, $mysqluser, $mysqlpasswd,
        $mysqldb, $mysqlport);
} catch (mysqli_sql_exception $e) {
  Error::display("Verbindungsfehler: ");
  switch ($e->getCode()) {
  case 1049:
    print "Die Datenbank $mysqldb konnte nicht gefunden werden.";
    print "Erstellen Sie die Datenbank mit ";
    print "<pre>CREATE DATABASE $mysqldb</pre>";
    break;
  case 1044:
  case 1045:
    printf("Benutzer %s verfügt nicht über ausreichende
      Berechtigungen", $mysqluser);
    printf("Verwenden Sie <pre>
      GRANT ALL PRIVILEGES ON %s.* TO %s@%s IDENTIFIED BY '%s';
      </pre>", $mysqldb, $mysqluser, $mysqlhost, $mysqlpasswd);
    break;
  case 2003:
    print "Es konnte keine Verbindung zum Server
      <i>$mysqlhost</i> hergestellt werden.";
    print "Stellen Sie sicher, dass der MySQL-Server auf dem Host
      $mysqlhost am Port $mysqlport aktiv ist";
    break;
  default:
    print $e->getMessage();
  }
  global $html;
  $html->printFooter();
  exit();
}
```

Tritt ein Fehler auf, der nicht explizit behandelt wird, so wird die Fehlermeldung der *mysqli_exception* ausgegeben (*default*-Abschnitt der *case*-Anweisung).

Das Beispielprogramm verwendet konsequent *prepared*-Statements bei den SQL-Abfragen. Die dadurch gewonnene Sicherheit beim Übergeben der Parameter hat ihren Preis beim Auswerten des Abfrage-Ergebnisses. Um die gleiche Funktionalität wie die schon bekannte *queryObjectArray*-Funktion zu erreichen (vergleiche Abschnitt 12.1.2), müssen ein paar Tricks angewendet werden:

```
function queryObjectArrayPrepared($stmt) {
  $fieldNames = array();
  $stmt->execute();
  $meta = $stmt->result_metadata();
  $fieldNames[0] = &$stmt;
  $myObject = new stdClass();
```

open source library

```
    foreach ($meta->fetch_fields() as $fields) {
      $fieldNames[$fields->name] = &$myObject->{$fields->name};
    }
    call_user_func_array("mysqli_stmt_bind_result", $fieldNames);
    $result = array();
    while ($stmt->fetch()) {
      $oRes = new stdClass();
      foreach ($fieldNames as $k=>$f) {
        if ($k === 0) {
          continue;
        }
        $oRes->$k = $f;
      }
      $result[] = $oRes;
    }
    return $result;
  }
```

Leider gibt es bei *prepared*-Statements nicht die Möglichkeit, das Ergebnis mit *fetch_object* abzuspeichern. Nach dem Ausführen des SQL-Kommandos (*$stmt->execute()*) muss die Ergebnis-Variable an das Statement gebunden werden. Normalerweise geschieht das mit *$stmt->bind_result()*, das Ergebnis beinhaltet dabei aber leider keine Spaltennamen. Diese Bezeichner sind in den Metadaten des Ergebnisses gespeichert und werden in die Array-Variable *$fieldNames* übertragen. Mit der *call_user_func_array*-Funktion ist es möglich, das so präparierte Array an das Ergebnis zu binden. Im weiteren Verlauf werden die Einträge Zeile für Zeile abgearbeitet (*while ($stmt->fetch()*), und es wird jeweils ein neues Objekt der Standard-Klasse erstellt, das mit den abgefragten Feldern gefüllt wird. Die Objekte werden anschließend in dem *$result*-Array gespeichert und zurückgegeben.

Eine weitere Besonderheit der *Db*-Klasse ist die statische Methode *getHandle()*. Sie wird verwendet, um zu verhindern, dass mehrere Verbindungen zur Datenbank von einem User hergestellt werden.

```
static function getHandle() {
  global $dbHandle;
  if (!$dbHandle instanceof Db) {
    $dbHandle = new Db();
  }
  return $dbHandle;
}
```

Da die unterschiedlichen Klassen unabhängig voneinander Zugriff auf die Datenbank brauchen, besteht die Notwendigkeit, die einmal hergestellte Datenbankverbindung allen zugänglich zu machen. Das Problem wird über den Trick einer globalen Variable (*$dbHandle*) gelöst. Ist diese bereits eine Instanz der *Db*-Klasse, wird sie unverändert zurückgegeben, andernfalls wird ein neues Objekt der *Db*-Klasse erzeugt. Im PHP-Code wird für den Zugriff auf die Datenbank immer diese Funktion aufgerufen, nie der Konstruktor der Klasse selbst.

18.4.2 Das iCalendar-Modul

iCalendar, oder kurz *iCal*, ist ein Internet-Standard (RFC 2445) zum Austausch von digitalen Kalender-Informationen. Im vorliegenden Beispiel wird das Format verwendet, um die Arbeitseinträge in einer Kalenderanwendung dynamisch einzubinden (siehe Abbildung 18.4). Das *ical*-Modul erzeugt dazu ein *Icalendar*-Objekt, das den Großteil der Arbeit übernimmt.

```
<?php //  Beispieldatei ical.php
$html->footLine = FALSE;
$id = $html->getIntFromRequest('id');
$tt = new Timetrack();
$ical = new Icalendar();
$ical->printHeader();
if ($id > 0) {
  $tasks = array(new TimetrackTask($id));
} else {
  $tasks = $tt->getTasks();
}
foreach ($tasks as $task) {
  $entries = $task->getEntries();
  if ($entries) {
    foreach ($entries as $entry) {
      $ical->addEvent($entry, $task->name);
    }
  }
}
$ical->printFooter();
```

Wird dem *ical*-Modul eine Projekt-ID übergeben, so werden nur die Arbeitseinheiten für das ausgewählte Projekt angezeigt. Andernfalls werden alle Einträge in den Kalender eingefügt. Nach der Ausgabe des *iCal*-Headers wird für jeden Eintrag die Funktion *addEvent* aus der *Icalendar*-Klasse aufgerufen. Ihr wird ein Objekt vom Typ *TimetrackEntry* und der Name des Projekts übergeben.

```
public function addEvent($entry, $name) {
  if ($this->headerPrinted === FALSE) {
    $this->printHeader();
  }
  print "BEGIN:VEVENT\n";
  print "DTSTART:" . DateFormatter::ical($entry->start) . "\n";
  print "DTEND:" . DateFormatter::ical($entry->end) . "\n";
  print "SUMMARY: " . $name . "\n";
  if ($entry->unfinished) {
    print "DESCRIPTION: still running\n";
  }
  print "END:VEVENT\n";
}
```

Die statische *ical*-Funktion in der *DateFormatter*-Klasse formatiert einen *UNIX_-TIMESTAMP* entsprechend den Vorgaben im iCalendar-Internet-Standard.

```
public static function ical($ts) {
  return date("Ymd\THi00", $ts);
}
```

Zum Beispiel wird der Zeitpunkt für den 18. Januar 1998 um 23:00 Uhr in einer iCalendar-Datei als 19980118T230000 dargestellt. Auf die zusätzliche Ausweisung der Zeitzone wurde in diesem Beispiel verzichtet. Einträge, die gerade aktiv geschaltet sind, werden mit dem zusätzlichen Feld *Description* versehen, das einen Hinweis auf die unvollendete Arbeitseinheit enthält.

18.4.3 Zeiteinträge bearbeiten

Das Bearbeiten der Zeiteinträge erfordert eine detaillierte Fehlerbehandlung: Die Endzeit darf nicht vor der Startzeit liegen, und die Pause darf nicht länger sein als die Gesamtdauer der Einheit. Außerdem dürfen nur gültige Datumswerte eingegeben werden. Wenn man die Überprüfung der Daten PHP überlässt, so kommt es bei jedem Versuch zu einem Request an den Server, und die Fehlermeldung wird zeitlich etwas verzögert zurückgegeben. Mit Javascript hat man die Möglichkeit, unmittelbarer einzugreifen. Da der Code im Webbrowser auf dem Client läuft, muss keine Verbindung zum Server hergestellt werden, und die Antwort kommt prompt.

Hier ein Auszug aus dem HTML-Quelltext der *editentry*-Seite (siehe Abbildung 18.3):

```
<form method="post" id="editEntryForm" action="editentry">
<div>
<table>
  <tr><td>Startzeit</td>
<td>
<input onchange="computeDuration()" size="2" maxlength="2"
name="from_DD" value="22" />
<input onchange="computeDuration()" size="2" maxlength="2"
name="from_MM" value="02" />
<input onchange="computeDuration()" size="4" maxlength="4"
name="from_YYYY" value="2008" />
...
```

Um auf die Felder eines HTML-Formulars zugreifen zu können gibt es in Javascript die Möglichkeit, eine Variable mit einer Referenz auf das Feld zu definieren. Das ist äußerst praktisch, da beim Aufruf der Variable immer der aktuelle Inhalt des Formularfeldes ausgegeben wird.

open source library

```
// datetime.js, Javascript-Bibliothek zum Korrigieren von Zeiteinträgen
var from = {};      // Startzeit Felder
var to = {};        // Endzeit Felder
var breakTime = 0;  // Dauer der Pause
var duration = 0;   // Gesamtzeit
var oForm = {};     // Referenz auf alle Formularfelder
function init() {
  oForm = document.forms.editEntryForm;
  from = _defineDateFields('from_');
  to = _defineDateFields('to_');
  computeDuration();
}
// Zuweisung der Feld-Referenzen
function _defineDateFields(prefix) {
  return {
    YYYY : oForm[prefix+'YYYY'],
    MM : oForm[prefix+'MM'],
    DD : oForm[prefix+'DD'],
    hh : oForm[prefix+'hh'],
    mm : oForm[prefix+'mm'],
    ts : oForm[prefix+'ts']
  }
}
```

Im *body*-Element des HTML-Dokuments wird der Browser mittels *onload* angewiesen, die *init*-Funktion auszuführen. Dort wird die Variable *oForm* mit einer Referenz auf das gesamte *FORM*-Element belegt. Anschließend werden die Variablen *from* und *to* mit den einzelnen Feldern des jeweiligen Start- und Enddatums belegt.

Wie *oForm* sind *from* und *to* Referenzen auf die Formularfelder und enthalten keine Kopie von deren Werten. Das ist wichtig, da beim mehrmaligen Verändern der Werte immer diese Variablen für die Berechnung herangezogen werden. Über den Aufruf von *oFrom.hh.value* hat man im Javascript-Code Zugriff auf den aktuellen Inhalt des Stunden-Feldes der Startzeit. Es macht Sinn, diese Referenzen nur einmal beim Laden der Seite zu initialisieren, da der Browser dabei spürbar Rechenleistung verbraucht. Die entscheidende Javascript-Funktion ist *computeDuration*, denn hier werden mögliche Fehler behandelt.

```
function computeDuration() {
  var dateFrom = new Date(from.YYYY.value,
    from.MM.value-1,from.DD.value,from.hh.value,from.mm.value);
  var dateTo = new Date(to.YYYY.value,
    to.MM.value-1,to.DD.value,to.hh.value,to.mm.value);
  // Prüfung der Daten
  if (isNaN(dateFrom)) {
    alert('Fehler: Startzeit ungültig');
    return false;
  }
```

```
    if (isNaN(dateTo)) {
      alert('Fehler: Endzeit ungültig');
      return false;
    }
    // Fehlerprüfung
    if (dateFrom.getTime() > dateTo.getTime()) {
      alert('Fehler: Startzeit muss vor Endzeit liegen');
      return false;
    }
    duration = (dateTo.getTime()-dateFrom.getTime())/1000/60;
    // Berechnung der Pausen-Zeit
    updateBreak();
    // Statusfelder aktualisieren
    document.getElementById('statusFrom').value = dateFrom
    document.getElementById('statusTo').value = dateTo
    updateDuration();
    from.ts.value = dateFrom.getTime()/1000;
    to.ts.value = dateTo.getTime()/1000;
  }
```

dateFrom und *dateTo* sind Javascript-Objekte vom Typ *Date* und repräsentieren einen Wert in Millisekunden nach dem 1.1.1970. Sie sind damit dem schon bekannten *UNIX_TIMESTAMP* ähnlich, nur dass ihre Genauigkeit tausendmal höher ist (was für das Zeiterfassungsbeispiel natürlich irrelevant ist).

Beim Initialisieren der Variable werden die Werte aus den Formularfeldern ausgelesen (zum Beispiel *from.YYYY.value* für den Jahreswert des Startdatums). Zu beachten ist, dass die Zählung der Monate in Javascript bei null beginnt, daher wird von der Formulareingabe immer ein Wert abgezogen (*from.MM.value-1*).

Mit der Funktion *isNaN* (Abkürzung für **is N**ot **a N**umber) wird überprüft, ob das Datumsobjekt einen korrekten Wert beinhaltet. Javascript ist dabei so großzügig, dass es einen zu hohen Wert automatisch konvertiert (zum Beispiel werden 99 Minuten automatisch zu 1 Stunde und 33 Minuten). Anschließend wird überprüft, ob es sich um eine gültige Zeitspanne handelt und ob die Pause nicht länger ist als die Gesamtdauer des Eintrags. Sollten alle Angaben richtig sein, werden die beiden Statusfelder (*statusFrom*, *statusTo*) aktualisiert und wird ein Textfeld mit der neuen Dauer befüllt (*updateDuration()*). Außerdem wird die gültige Beginn- und Endzeit in je einem versteckten Eingabefeld (*from_ts*, *to_ts*) als *UNIX_TIMESTAMP* mitgeführt. Im PHP-Code entfällt dadurch ein neuerliches Zusammensetzen der Datumsfelder.

```
  if ($html->getStringFromRequest("save")) {
    $entry->note = $html->getStringFromRequest("note");
    $entry->start = $html->getIntFromRequest("from_ts");
    $entry->end = $html->getIntFromRequest("to_ts");
    $entry->removeTime = $html->getIntFromRequest("breakTime");
    $entry->save();
```

open source library

```
    print "<p>Erfolgreich gespeichert. ";
    printf("<a href='view?id=%d'>zurück zur Liste</a>", $entry->task_id);
    print "</p>\n";
  }
```

18.4.4 Die DOMList-Klasse

Die *DOMList*-Klasse wird verwendet, um die Projekte in unterschiedlichen Formaten auszugeben: einmal als HTML-Liste, dann als HTML-Dropdown-Menü und schließlich noch als XML-Dokument für die Verwendung in der Zeitlinie. Dabei wird jeweils ein *DOMDocument* erzeugt und als XML-Fragment ausgegeben.

Bei der HTML-Liste (eine *unorderd list*, siehe Abbildung 18.8) wird als Erstes ein UL-Element erzeugt, dem die ID 0 zugewiesen wird. Über die Funktion *getElementById* wird in der Schleife über alle Projekte das übergeordnete Listenelement gesucht (*$task->parentId*). An dieses Element wird anschließend ein neues *li*-Element angehängt (*$parent->appendChild*).

```
function generateUl($tasks) {
  $this->addUlElement(0, $this->dom);
  foreach ($tasks as $task) {
    $parent = $this->dom->getElementById("dbid_".$task->parentId);
    if (!$parent) {
      continue;
    }
    $li = $parent->appendChild($this->dom->createElement('li',
      $task->name." (".
      DateFormatter::formatDuration($task->totalTime).") "));
    $this->addLink($task->id, $li, 'view', 'anzeigen');
    $this->addLink($task->id, $li, 'edittask', 'bearbeiten');
    if ($task->hasSubtree()) {
      // valid XHTML1 needs one empty 'li' Element for each ul Element
      $emptyLi = $parent->appendChild($this->dom->createElement('li'));
      $emptyLi->setAttribute('style', 'list-style:none');
      $this->addUlElement($task->id, $emptyLi);
    }
  }
}
```

Die Funktion *addUlElement* wird nur dann aufgerufen, wenn das Projekt noch weitere Unterprojekte besitzt. In ihr wird das für den *getElementById*-Aufruf benötigte ID-Attribut gesetzt:

```
function addUlElement($id, $parentElement) {
  $ulChild = $parentElement->appendChild($this->dom->createElement('ul'));
  $ulChild->setAttribute('id', "dbid_".$id);
  $ulChild->setIdAttribute('id', true);
}
```

Durch das Überschreiben der *__toString*-Funktion kann das *DOMDocument* anschließend einfach mit *print* ausgegeben werden. Die Anweisung *formatOutput = true* bewirkt, dass der XML-Baum mit Einrückungen formatiert ausgegeben wird.

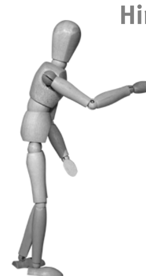
```
function __toString() {
  $this->dom->formatOutput = true;
  return $this->dom->saveXML($this->dom->documentElement);
}
```

Das Modul zum Anzeigen der HTML-Liste beschränkt sich auf wenige Zeilen Code:

```
<?php   // task.php-Modul zum Anzeigen aller Projekte
$html->printHeader('Liste aller Projekte');
?>
<h2>Projekte</h2>
<p><a href='addTask?p_id=0'>Projekt auf oberster Ebene hinzufügen</a></p>
<?php
$tt = new TimeTrack();
$dom = new DOMList();
$dom->generateUl($tt->getTasks());
print $dom;
```

18.4.5 Der Timeline-Code

Bei der Zeitlinie aus dem SIMILE-Projekt handelt es sich um eine moderne Javascript-Bibliothek, die zeitbezogene Events grafisch ansprechend darstellt. Sie ist sehr einfach zu verwenden und bietet eine erstaunliche Interaktivität.



Hinweis

Auf der Webseite des SIMILE-Projekts finden Sie noch weitere Open Source-Bibliotheken und -Applikationen mit dem Schwerpunkt Visualisierung und Konvertierung von XML-Dokumenten. Die in diesem Beispiel verwendete *timeline*-Bibliothek beeindruckt durch das geschmeidige Verschieben der Zeitachse bei gedrückter Maustaste. Weitere Beispiele, die noch mehr Möglichkeiten dieser Bibliothek zeigen, finden Sie auf der Webseite des Projekts:

http://simile.mit.edu/timeline/

Im HTML-Header werden zwei Javascript-Dateien geladen: zuerst die Bibliothek selbst, die direkt vom Server des MIT kommt, und dann eine lokale Datei, in der die Anwendung konfiguriert wird.

```
<script src="http://simile.mit.edu/timeline/api/timeline-api.js"
  type="text/javascript"></script>
<script src="script/timeline.js" type="text/javascript"></script>
```

open source library

Die *onload*-Anweisung im Body-Element der HTML-Seite startet die *onLoad*-Funktion in der lokalen Datei timeline.js und veranlasst die Initialisierung der Grafik.

```
<body onload="onLoad();" onresize="onResize();">
<h2>Zeitlinie</h2>
<div id="timetracktimeline"
style="height:300px;width:100%;border:1px solid #aaa"></div>
```

Die Zeitlinie findet in einem *div*-Element Platz. Über das *id*-Attribut (hier *timetrack-timeline*) wird sie aus Javascript angesteuert.

Der interessante Teil bei diesem Modul liegt in der Javascript-Datei timeline.js. Hier werden die Einstellungen für die Grafik festgelegt: Anzahl und Aussehen der Zeitleisten (Bänder) und die Quelle für die darzustellenden Daten.

```
// timeline.js Javascript-Bibliothek zur Steuerung der timeline
var tl;
function onLoad() {
  var eventSource = new Timeline.DefaultEventSource();
  var bandInfos = [
    Timeline.createBandInfo({
        eventSource:    eventSource,
        trackGap:       0.2,
        width:          "70%",
        intervalUnit:   Timeline.DateTime.HOUR,
        intervalPixels: 50
    }),
    Timeline.createBandInfo({
        eventSource:    eventSource,
        showEventText:  false,
        trackHeight:    0.5,
        trackGap:       0.2,
        width:          "20%",
        intervalUnit:   Timeline.DateTime.DAY,
        intervalPixels: 100
    }),
...
```

Das Javascript-Array *bandInfos* enthält die Definition der Zeitleisten. Das erste Band belegt 70 Prozent der gesamten Leiste und zeigt Stunden an, während das zweite Band ein Tagesintervall darstellt und 20 Prozent belegt. Um die Zeitleisten beim Verschieben synchron zu halten, bedarf es einer entsprechenden Anweisung:

```
bandInfos[1].syncWith = 0;
bandInfos[2].syncWith = 0;
```

Sowohl Band eins als auch Band zwei synchronisieren sich mit dem Band 0. Nach diesen Einstellungen kann die Zeitleiste erzeugt werden:

```
tl = Timeline.create(document.getElementById("timetracktimeline"),
    bandInfos);
Timeline.loadXML("xmldata", function(xml, url) {
    eventSource.loadXML(xml, url); });
```

Die *loadXML*-Funktion erwartet als ersten Parameter die Adresse, an der die XML-Daten abgerufen werden können. Für die Javascript-Bibliothek ist xmldata hier ein relativer Pfad am Webserver. Da es weder ein Verzeichnis noch eine Datei xmldata gibt, kommt wieder der Front-Controller zum Einsatz. Er sucht nach dem Modul xmldata und führt es aus.

```
<?php   // xmldata.php
header("Content-Type: text/xml");
$html->footLine = FALSE;
$timetrack = new Timetrack();
$dom = new DOMList();
$dom->generateDataList($timetrack->getTasks());
print $dom;
```

Damit das XML-Dokument korrekt ausgeliefert wird, muss der HTTP-Header *Content-Type* entsprechend gesetzt werden. Weiters wird der HTML-Abschluss (*$html->footline*) unterdrückt, den der Front-Controller normalerweise ausgibt.

18.4.6 Protokollieren der Aktivität

Wie schon eingangs erwähnt wurde, besitzt das Programm eine Möglichkeit, die Online-Aktivität teilweise zu protokollieren. Dabei wird in bestimmten Zeitintervallen ein *Timestamp*-Feld in der Datenbank mit der aktuellen Uhrzeit befüllt. Diese Funktion wird von einem PHP-Skript ausgeführt, das für den Benutzer unsichtbar, sozusagen im Hintergrund aufgerufen wird.

XMLHttpRequest

Wieder kommt hier Javascript zum Einsatz, wobei der asynchrone *XMLHttpRequest* für das Update in der Datenbank sorgt. Diese Javascript-Funktion führt eine HTTP-Abfrage aus, ohne dabei die Seite neu zu laden. Die Datenübertragung erfolgt im Hintergrund, und das Ergebnis kann mittels Javascript weiterverarbeitet werden.

Leider ist die Implementierung des *XMLHttpRequest*-Objekts in älteren Browsern stark unterschiedlich, weshalb der erste Teil des Codes sich nur damit beschäftigt, es allen Browsern recht zu machen. Inzwischen gibt es einen Standardisierungsvorschlag vom World Wide Web Consortium, der die Schnittstelle für einen einheitlichen Zugriff definiert.

open source library

```
<script type="text/javascript">
var xmlHttp = null;
// lade XMLHttpRequest-Objekt
// (siehe http://de.wikipedia.org/wiki/XMLHttpRequest)
// moderne Browser:
if (typeof XMLHttpRequest != 'undefined') {
  xmlHttp = new XMLHttpRequest();
}
if (!xmlHttp) {
  // IE 6 und älter
  try {
    xmlHttp  = new ActiveXObject("Msxml2.XMLHTTP");
  } catch(e) {
    try {
      xmlHttp  = new ActiveXObject("Microsoft.XMLHTTP");
    } catch(e) {
      xmlHttp  = null;
    }
  }
}
function setAlive() {
  if (xmlHttp) {
    xmlHttp.open('GET', 'setalive?id=$taskId', true);
    xmlHttp.onreadystatechange = function () {
      if (xmlHttp.readyState == 4) {
        statusDiv = document.getElementById('statusdiv');
        statusDiv.innerHTML = xmlHttp.responseText;
        // alert('debug: '+xmlHttp.responseText);
      }
    };
    setTimeout('setAlive()', 180000);  // millisecs, 180000 -> 3mins
    xmlHttp.send(null);
  }
}
setAlive();
</script>
```

Nachdem das *xmlHttp*-Objekt erzeugt worden ist, kommt die *setAlive*-Funktion zum Einsatz. Mit einem *HTTP-GET*-Aufruf wird die Adresse *setalive?id=$taskId* abgefragt. Da die Anfrage im Hintergrund läuft, ist die Auswertung des Ergebnisses etwas aufwendiger. Das *xmlHttp.onreadystatechange*-Attribut wird mit einer Funktion belegt, die auf jede Änderung im Ablauf der Abfrage reagiert. Enthält die Variable *readyState* den Wert 4, so ist der Request abgeschlossen und vollständig übertragen. Daraufhin wird der *responseText*, also die Antwort des PHP-Skripts auf der Webseite im *div*-Bereich mit der ID *statusdiv* angezeigt. Der Browser lädt dabei keine neue Seite: Für den Anwender wird der Text wie von Geisterhand ausgetauscht.

Die *setTimeout*-Funktion in Javascript verzögert den Aufruf der ihr übergebenen Funktion um eine gewisse Zeitspanne. Als Parameter werden einerseits der Name der Funktion und andererseits die Dauer der Verzögerung in Millisekunden angegeben. Durch den rekursiven Aufruf von *setAlive* läuft der Javascript Code so lange, bis der Browser die Seite verlässt.

Online-Status in der Datenbank aktualisieren

Nachdem der Javascript-Code regelmäßig Anfragen an das Modul *setAlive* stellt, muss dieses Skript nur noch den entsprechenden Eintrag in der Datenbank aktualisieren. Dazu wird zuerst das *lastseen*-Feld in der Tabelle aktualisiert, und anschließend werden die Startzeit und das soeben aktualisierte Feld abgefragt.

```
$sql = "UPDATE timelog SET lastseen = NOW() WHERE endtime IS NULL "
    . " AND task_id = ?";
$dbHandle = Db::getHandle();
$stmt = $dbHandle->prepare($sql);
$stmt->bind_param("i", $id);
$stmt->execute();
$sqlStatus = "SELECT * FROM timelog WHERE "
    . " endtime IS NULL AND task_id = ?";
$stmt = $dbHandle->prepare($sqlStatus);
$stmt->bind_param("i", $id);
$res = $dbHandle->queryObjectArrayPrepared($stmt);
if (! $res) {
  Error::display('No active task');
  exit();
}
print "<p>eingeloggt seit: " . $res[0]->starttime;
print "<br />letztes update: " . $res[0]->lastseen;
```

Die mit *print* ausgegebenen Zeilen erhält der *XMLHttpRequest* als *responseText*, der mithilfe von Javascript in den *statusdiv* eingesetzt wird.

18.5 Erweiterungsmöglichkeiten

Die Anforderungen an ein digitales Arbeitszeiterfassungssystem können sehr breit gestreut sein. Angefangen vom einfachen Tabellenkalkulationsblatt bis hin zur integrierten Projektverwaltung mit Stundenbudgets, Milestones und automatischer Verrechnung – der Phantasie sind hier keine Grenzen gesetzt. Dementsprechend kann dieses Beispiel noch in viele Richtungen erweitert werden:

- **Mehrbenutzerbetrieb:** In der vorliegenden Form ist das System nur für eine Person verwendbar. Das Datenbanksystem bietet aber geradezu ideale Voraussetzungen dafür, es nicht darauf zu beschränken. In einem Unternehmen würde man idealerweise an eine bestehende Benutzerverwaltung andocken (zum Beispiel über LDAP, vergleiche Abschnitt 5.2.2). Die Datenbankstruktur müsste für jeden Eintrag eine Verknüpfung zum Besitzer vorsehen, während der PHP-

open source library

Code sich darum kümmert, dass jeder Benutzer nur seine Einträge sehen und editieren darf. Weiters wäre eine Auswertung der gearbeiteten Stunden nach einzelnen Benutzern denkbar.

- **Projektverwaltung:** Die hier vorgestellte Form der Projektverwaltung ist äußerst rudimentär. Bei größeren Projekten gibt es meist einen Zeitplan und einen Budgetplan. Gekoppelt an die Anzahl der bereits gearbeiteten Stunden könnten Hinweise und Warnungen erscheinen, wenn sich das Stundenbudget dem Ende zuneigt. In Verbindung mit dem Mehrbenutzerbetrieb können einzelne Mitarbeiter bestimmten Projekten zugewiesen werden. Benutzer können dann ihre eigenen, privaten Projekte bearbeiten und an verteilten Projekten mit mehreren Benutzern teilnehmen.
- **Anbindung an Kundendaten:** Bei einer erweiterten Projektverwaltung wäre es denkbar, Daten zum Auftraggeber zu speichern. Für eine spätere Auswertung dieser Daten sind der Phantasie keine Grenzen gesetzt (Stichwort: *Customer Relationship Management*-Software).
- **Automatische Rechnungsstellung:** Die Stundenlisten könnte man um eine finanzielle Bewertung der Tätigkeit aufwerten (zum Beispiel kann für die Tätigkeit der Systemadministration ein anderer Stundenlohn veranschlagt werden als für die Programmierung). Aus der Gesamtsumme der Stundenlisten lässt sich der Rechnungsbetrag für das Projekt erstellen. Durch die Anbindung der Adressdaten der Kunden kann die Rechnung automatisch erstellt werden. Im wirklichen Leben wird es hier aber häufig einer manuellen Nachbearbeitung bedürfen (Stichwort: *Skonto*).

18.6 Installation

Da das Beispiel auf die Trigger-Funktionen von MySQL zurückgreift, müssen Sie mindestens MySQL 5.0.3 einsetzen. Der PHP-Code verwendet die *mysqli*-Erweiterung mit der Unterstützung von *prepared*-Statements, was ab PHP 5.2.1 per Default aktiviert ist.

- **Datenbank erzeugen:** Zu Beginn erstellen Sie eine leere Datenbank mit dem Namen *timetrack*. Dazu können Sie phpMyAdmin verwenden oder direkt im MySQL-Kommandozeileninterpreter *CREATE DATABASE timetrack* ausführen.
- **Tabellen erzeugen:** Ein Dump der Tabellenstruktur liegt in der Datei `tt_tables.sql` im Verzeichnis `maintenance`. Einspielen lässt sich diese Datei wieder unter phpMyAdmin oder auf der Kommandozeile mit *mysql timetrack < tt_tables.sql*.
- **PHP-Dateien kopieren:** Kopieren Sie im nächsten Schritt den gesamten Verzeichnisbaum `timetrack` in ein Verzeichnis, auf das der Webserver zugreifen kann.

- **Konfiguration anpassen:** Die Datei `config.inc.php` enthält Definitionen für einige Standardeinstellungen (wie viele Ergebnisse werden pro Seite angezeigt, ab welcher Dauer einer Arbeitseinheit wird eine Warnung angezeigt, ...). Wer mit den Voreinstellungen nicht zufrieden ist oder eine spezielle Konfiguration bei dem Webserver verwendet (unterschiedlicher Port oder *https* statt *http*), der kann hier Anpassungen vornehmen.

  Durch die ausführlichen Fehlermeldungen in der *Db*-Klasse sollte sich ein möglicher Fehler schnell finden und beheben lassen. Im Produktionsbetrieb sollte das Passwort nicht mehr ausgegeben werden.